An Elementary Algorithm for Reporting Intersections of Red/Blue Curve Segments*

Jean-Daniel Boissonnat† Antoine Vigneron‡

Abstract

Let E_r and E_b be two sets of non-intersecting curve segments, let $E = E_r \cup E_b$. When $|E| = n$, we give a new sweep-line algorithm algorithm that reports the k intersecting pairs of segments of E. Its time complexity is $O((n+k)\log n)$, it requires $O(n)$ space and makes use of simple primitive operations.

1 Introduction

The usual model to analyze geometric algorithms is the Real RAM which is assumed to compute exactly with real numbers [13]. This model hides the fact that the arithmetic of real computers has a limited precision and ignores numerical and robustness issues. As a consequence a direct implementation of an algorithm that is correct under the Real RAM model does not necessarily translate into a robust and/or efficient program, and catastrophic behaviours are commonly observed.

A first approach to remedy this problem is to use exact arithmetic. In the context of geometric algorithms, much progress has been done in the recent past [16, 17, 14, 12]. Another approach, to be followed here, has emerged recently. Decisions in geometric algorithms depend on geometric predicates which are usually algebraic expressions. For example, for a triple of points given by their cartesian coordinates, deciding what is the orientation of the triangle reduces to evaluating (the sign of) a multivariate polynomial of degree two. If an algorithm only uses predicates of degree 2 as a function of the input data, and if the input data are coded as simple fixed precision numbers, computations can be done exactly using the native double precision hardware of the computer. The degree of an algorithm is therefore related to the precision required to run an algorithm using exact arithmetic. This motivates the design of efficient algorithms of low degree. Reducing the degree of the algorithms will reduce the number and the complexity of the degenerate configurations, make the algorithms more elementary and more general, reduce the amount of numerical computations, which is usually quite a large fraction of the total execution time especially if multi-precision computing is invoked, and possibly also refrain the use of complicated data structures resulting in low space requirements. However, reducing the degree of an algorithm may increase its time complexity in the Real RAM model. Following Liotta et al. [11], we consider the degree of the predicates as an additional measure of the complexity of problems and algorithms, and intend to elucidate the relationship between time-complexity and degree of the predicates. Related research can be found in Knuth's seminal work [10] and in some recent papers [8, 4, 3].

In this paper, we consider the problem of reporting the k intersecting pairs among a set of n monotone curve segments. We address the red/blue case, where this set is partitioned into two subset of non intersecting segments. This problem can be solved in optimal $O(n \log n + k)$ time [5, 7, 1]. However these algorithms use predicates of high degree, e.g. to compare the abscissae of two intersection points or to locate an intersection point with respect to a vertical slab. These predicates have a degree and an algebraic complexity that are usually higher than the intersection predicate : this is in particular the case for line segments and circle segments [3]. Our algorithms only use the intersection predicate and two other simple predicates : the predicate that sorts two points by abscissae, and the predicate that says if a point is below, on, or above a segment. In particular, we do not compute the arrangement nor the trapezoidal map of the segments. Moreover, the predicates we use do not say anything about the number or the positions of the intersection points, and the time complexity of these algorithms depends only on the number of intersecting pairs of segments, not the number of intersection points (differently from the other non trivial algorithms [2, 7, 6, 5, 1]).

The time complexity of our algorithm is $O((n+k)\log n)$, which is a log factor from optimal. This result generalizes a similar result for the case of pseudo-segments, i.e. segments that intersect in at most one point [3].

*This work has been partially supported by the ESPRIT IV LTR Project No 28155 (GALIA) and the Action de Recherche Coopérative Géométrica.

†INRIA Sophia Antipolis, 2004 Route des Lucioles, BP 93, 06904 Sophia-Antipolis, France.

‡Hong Kong University of Science and Technology.

2 The problem

A curve segment is *monotone* if it is the graph of a partially defined univariate function (i.e. any vertical line intersects such a segment in at most one point). Let $E = E_r \cup E_b$ be a set of n monotone curve segments such that no two segments of E_r (resp. E_b) intersect. The problem is to report the k pairs of segments of E that intersect.

Let s and s' be two segments of E and let p and q be two endpoints of some segments of E, not necessarily of s or s'. $x(p)$ and $y(p)$ denote the coordinates of p, and $s(x)$ denotes the point of s whose abscissa is x (if such a point exists). We consider the following predicates :

Predicate 1: $s \cap s' \neq \emptyset$
Predicate 2: $x(p) \leq x(q)$
Predicate 3: $y(p) \leq y(s(x(p)))$
Predicate 4: $y(s(x(p))) \leq y(s'(x(p)))$
Predicate 5: $\exists x \in [x(p), x(q)]$ such that $s(x) = s'(x)$

Predicate 1 is mandatory. Predicate 2 allows to sort the endpoints. Predicate 3 tells whether an endpoint lies above or below a segment. Predicate 4 provides the order of two segments along a vertical line passing through an endpoint. Predicate 5 checks whether two segments intersect within a vertical slab defined by two endpoints.

We do not specify a precise representation for the segments, which may depend on the application. For example, the function associated to a segment may be given explicitly together with the interval where it is defined, or the function may be given implicitly as the algebraic function of degree d that interpolates $d+1$ given points (including the endpoints of the segment). Other representations are also possible. The degrees of the predicates clearly depend on the chosen representation. In the table below (see also [3, 9]), the degrees of Predicates 2-5 are given for line segments represented by the coordinates of their endpoints, for half-circles defined by centers, radii plus a boolean (to distinguish between the upper and the lower arc), for circle segments defined by three points (including the two endpoints of the segment), and for curves defined by a polynomial equation $y = f(x)$. Observe that the predicates are ordered by increasing degrees in all those cases. However, this order may be different for some other types of segments. For instance, for half-circles defined by three non x-extreme points, the degree of Predicate 2 is 20 while the degree of Predicate 3 is only 4.

Predicate	degree			
	line segment	half circle	circle segment 3 points	pol. of degree d
1	2	2	12	d
2	1	1	1	1
3	2	2	4	d
4,5	3	4	12	d

Our algorithm only requires predicates 1,2 and 3 while other non-trivial algorithms [2, 7, 6, 5, 1]) make use of our five predicates. Thus they have higher algebraic degree when we consider line segments, half circles or circles defined by three points.

3 Algorithm

In this section, e_b (resp. e_r) denotes a blue (resp. red) segment, that is $e_b \in E_b$ and $e_r \in E_r$. We will prove the following result:

Theorem 1 *The red-blue curve segment intersection problem can be solved in $O((n+k)\log n)$ time and $O(n)$ space using predicates 1, 2 and 3.*

We present a new plane sweep algorithm for this problem. A vertical sweep line moves across the plane from left to right. When it reaches an enpoint, some data structures are updated and intersections can be reported. At any time, we only consider *active* segments which are the segments that cross the sweep line.

Definition 1 *A good intersection (see figure 1) is an intersection between a blue segment e_b and a red segment e_r such that the left endpoint of e_b is above e_r and on the right of the left endpoint of e_r.*

In the following subsections, we describe an algorithm that reports the *good* intersections. The other intersections are reported by the same algorithm after exchanging the orientation of the y-axis or the colors of the segments. The algorithm to be described is therefore applied four times.

3.1 Data structures

The idea is to report the good intersections while pushing the blue segments downwards. Each blue segment is stored in a set $U(e_r)$ for some active red segment e_r. The set of segments in $U(e_r)$ will be maintained during the sweep. It consists of the blue segments that cannot be pushed down because they are blocked either by e_r or by the lowest segment of $U(e_r)$.

The above/below relationship within E_b (resp. E_r) is a partial order, which is a total order when restricted to active segments. We will maintain the ordered list of the active red segments in a balanced search tree just like in Chan's algorithm [5] for red-blue segments.

For each blue segment e_b, we associate a red segment $l(e_b)$. We will require that the two following properties remain satisfied during the course of the algorithm :

Property 1 *The good intersections of e_b with a red segment that lies above $l(e_b)$ have been reported previously (see figure 3).*

Property 2 *As the plane is swept, $l(e_b)$ decreases with respect to the vertical order of red segments.*

For each active red segment e_r, we denote by $U(e_r)$ the set of all active blue segments e_b such that $l(e_b) = e_r$. It is stored in a data structure that allows union, minimum extraction and suppression in $O(\log n)$ time (e.g. a binomial heap). The underlying order is the vertical order of active red segments along the sweep line. Note that we shall not maintain $l(e_b)$ explicitely, as it is not clear whether we can do it within the same time bounds. Its value can be retrieved in logarithmic time from the heaps $U(e_r)$. For any red segment e_r, the set $U(e_r)$ satisfies the following property:

Property 3 *The minimum of $U(e_r)$ does not intersect e_r. The other segments of $U(e_r)$ can only intersect e_r or any red segment below e_r on the right of the right endpoint of this minimum (see figure 1).*

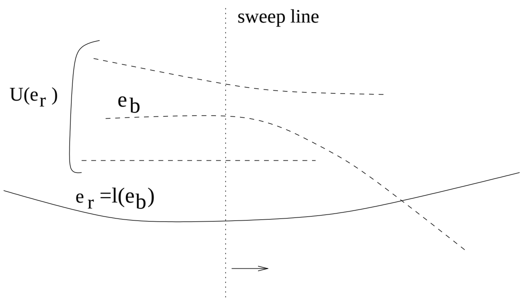

Figure 1: $U(e_r)$ and a good intersection

However, the sets $U(e_r)$ do not have any simple monotonicity property. For example, if e_r is below e'_r, then $U(e_r)$ may be entirely above $U(e'_r)$ or they may be interlaced (see figure 2).

3.2 Handling the events

We have to consider four kinds of events that occur when the sweep line reaches an endpoint of a segment. Each event corresponds to the insertion, the deletion of a blue or a red segment. We will now explain in detail how these events are handled, and prove that our invariants (properties 1, 2, 3) are maintained.

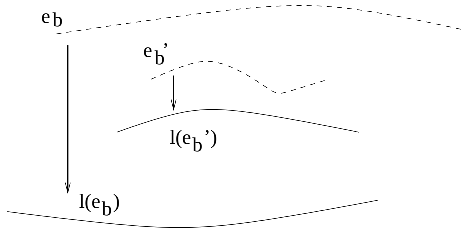

Figure 2: $U(e_r)$ is above $U(e'_r)$.

3.2.1 Inserting a red segment

The new red segment e_r is just inserted in the list of active red segments and the heap $U(e_r)$ is initialized as an empty set. Our invariants are obviously maintained.

3.2.2 Inserting a blue segment

When the sweep line reaches the endpoint of a blue segment e_b, we first determine the red segment e_r that lies just below the endpoint. If e_b does not intersect e_r, then it is inserted in $U(e_r)$. Otherwise, we report this intersecting pair and repeat the same process with the active red segment that is below e_r until we reach a red segment that does intersect e_r .

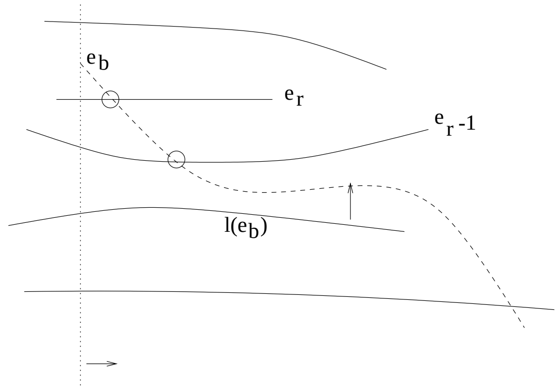

Figure 3: Insertion of e_b

This procedure can be written in the following way, where e_{r-1} denotes the active red segment that lies immediately below e_r.

Insert(e_b, e_r)
While e_b intersects e_r
    report the intersection
    $e_r \leftarrow e_r - 1$
Insert e_b in $U(e_r)$

The intersections between e_b and the red segments that are below the left endpoint of e_b and above $l(e_b)$

have been reported, therefore property 1 is true for e_b. Besides, it is still valid for the other blue segments.

Since $l(e)$ remains the same for each active blue segment e, property 2 still holds after the insertion of e_b.

Finally, let us check property 3. If e_b is the new minimum of $U(e_r)$ where $e_r = l(e_b)$, then of course it does not cross $l(e_b)$. Moreover, the other segments of $U(e_r)$ may only cross $U(e_r)$ on the right of the sweep line, therefore they can not cross it before the right endpoint of e_b is reached since e_b lies between these segments and e_r. The case where e_b is not the new minimum of $U(e_r)$ is trivial.

3.2.3 Removing a red segment

Now suppose the sweep line reaches the right endpoint of a red segment e_r. m denotes the minimum of $U(e_r)$. If m intersects $e_r - 1$, then report the intersection, try again with $e_r - 2$ ... until a segment $e_r - j$ is found that does not intersect m. Then m is moved from $U(e_r)$ to $U(e_r - j)$, and we start again with the new minimum of $U(e_r)$. Otherwise, if m does not intersect e_r, we simply merge $U(e_r)$ and $U(e_r - 1)$ thus maintaining property 3.

In this algorithm, **Insert** is the procedure we described in the previous section.

```
While m = min(U(e_r)) intersects e_r − 1
        report the intersection
        extract m from U(e_r)
        Insert(m,e_r − 2)
U(e_r − 1) = U(e_r − 1) ∪ U(e_r)
```

Bad intersections may be reported during this operation, we may just discard them by comparing the coordinates of the left endpoints.

Note that $l(e_b)$ is not updated after merging $U(e_r - 1)$ and $U(e_r)$ in order to keep our time complexity bound.

The invariants are maintained as is easily checked as in the previous case.

3.2.4 Removing a blue segment

Let e_b be the segment to be removed and $e_r = l(e_b)$. As we said before, $l(e_b)$ can be obtained in $O(\log n)$ time from the mergeable heap data structure that stores the sets $U(.)$.

If e_b is not the minimum of $U(e_r)$, we just remove it. Otherwise, we still remove e_b from $U(e_r)$, but then we need to check whether the new minimum m of $U(e_r)$ intersects e_r. If it does, we push it downwards by running the procedure **Insert**$(m, e_r - 1)$, then repeat the whole process with the new minimum until it does not intersect e_r, so that property 3 remains true.

```
extract e_b from U(e_r)
while m = min(U(e_r)) instersects e_r
        report this intersection
        extract m from U(e_r)
        Insert(m,e_r − 1)
```

Once again, this procedure may report bad intersection which can be fixed by a simple test.

It can be easily checked that the invariants are maintained.

3.3 Proof

The correctness of this algorithm directly follows from properties 1 and 3.

3.4 Analysis

Maintaining the ordered red segments list takes $O(n \log n)$ time. Localizing an endpoint or removing it takes $O(\log n)$. The other parts of the algorithm take at most $O(\log n)$ for each event or reported intersection, then we just need to prove that an intersection cannot be reported twice. Let e_r and e_b be two intersecting segments. After reporting their intersection once, $l(e_b) < e_r$. Moreover, property 2 means that $l(e_b)$ can only go down the list of red active segments, and we never test the intersection between e_b and a segment that lies above $l(e_b)$.

3.5 Degenerate cases

Since this algorithm is elementary we only need to consider a few degenerate cases which turn out to be very easy to handle. Two kinds of degeneracy may occur: either two endpoints have the same abcissa or an endpoint lies on a curve.

The first case can be solved by extending the partial order on endpoints abcissae to any total order. The order should be the same for each one of the four plane sweeps we perform for otherwise some intersections would never be good.

If a point lies on a segment, we just consider that it is above the segment during two sweeps and below the segment during the two sweeps with the y-axis reversed.

4 Conclusion

Given a set of general monotone segments, we have presented algorithms to report the pairs of segments that intersect. Our algorithms use only three simple predicates that allow to decide if two segments intersect, if a point is left or right to another point, and if a point is above, below or on a segment. These three predicates seem to be the simplest predicates that lead to subquadratic algorithms. Our algorithms are almost optimal in this restricted model of computation.

We conclude with some open problems. First, can we remove a $\log n$ factor in our time complexity results ?

The $\Omega(n\sqrt{k} + n \log n)$ lower bound holds for general curve segments. It has been possible to do better for line segments [3]. Can we also do better for other special curve segments such as circle segments. Some preliminary results for thick objects have been obtained by Vigneron [15].

In this paper, we have restricted our attention to monotone segments. This may be a restriction when the points with a vertical tangent are difficult to compute. This is in particular the case for circles defined by three points where the predicate that compares x-extreme points has degree 20 while the intersection predicate has degree 12 only.

References

[1] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[3] J.-D. Boissonnat and J. Snoeyink. Efficient algorithms for line and curve segment intersection using restricted predicates. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 370–379, 1999.

[4] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comput.* to appear.

[5] T. M. Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 263–268, 1994.

[6] Bernard Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[7] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[8] F. d'Amore, P. G. Franciosa, and G. Liotta. A robust region approach to the computation of geometric graphs. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms – ESA '98*, volume 1461 of *Lecture Notes Comput. Sci.*, pages 175–186. Springer-Verlag, 1998.

[9] Olivier Devillers, Alexandra Fronville, Bernard Mourrain, and Monique Teillaud. Exact predicates for circle arcs arrangements. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 139–147, 2000.

[10] Donald E. Knuth. *Axioms and Hulls*, volume 606 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, Germany, 1992.

[11] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[12] Sylvain Pion. Interval arithmetic: an efficient implementation and an application to computational geometry. In *Workshop on Applications of Interval Analysis to systems and Control*, pages 99–110, 1999.

[13] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, October 1990.

[14] S. Schirra. Robustness Issues *Handbook of Computational Geometry*. Elsevier Science Publishers B.V. North-Holland, 1999.

[15] A. Vigneron. Algorithmes élémentaires pour reporter les intersections d'objets courbes. Rapport de DEA algorithmique, Paris, France, 1999.

[16] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.

[17] C. K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997.